Sprint 3 – Guidelines

Deadline: Oct 4th , 2020 23:59
Definition of Done: Sep 24th, 2020 16:00

TAs: Gunnar Jörgen Viggósson gunnarv11@ru.is (Reykjavík)
Svanur Jóhannesson svanurjoh@ru.is (Akureyri)
Þórður Friðriksson thordurf@ru.is (Reykjavík)

Here is a list of aspects that will be considered when marking this Sprint.

Important

- No UI is required
- There is a peer-review assessment at the end of each Sprint

Even more important

- Teams need to **pass 4 out the 5** Sprints in order to pass the Project component of the course
- We are grading your project based on the last commit before the deadline!!

Grading Aspects

These are only guiding aspects; grading will not be limited to the description below. Be thorough!

1. **Definition of done.** You have agreed with the Product Owner (PO) a definition of done. Asana but also communicate this to Gunnar and Svanur via email, Tuesday Sep 1st at the latest.
2. **Sprint planning.** The sprint has been planned – user stories have been suggested to the TA. Asana
3. **Task breakdown.** User stories have been broken down into tasks by the group and assigned to group members. Asana
4. **Test coverage.** Statement coverage (only the application folder, not counting tests or external libraries). Less than 50% is not acceptable neither just 51%. Unittest suffices.
5. **Comments / readability.** Quality comments, indicate where are you addressing Lecture Aspects.
6. **Standup / decision protocol.** Is the team organized, getting together to discuss the Sprint tasks? Asana.
7. **Retrospective /reflection.** There is clear evidence from the retrospective: what worked, what didn’t, team collaboration. Asana.

8. **Technical requirements**
9. **README.md** file included
10. **Lecture Aspects**
    a. Hand in Context, Container, Component and Class diagrams (Visualizing Architecture lecture). One delivery diagram, one context diagram, two container diagrams, four component diagrams and four class diagrams.
    b. Identify or implement, at least three places, where you are using the Single Responsibility Principle (Visualizing Architecture lecture)
    c. Identify or implement, at least one, Dependency Injection (Inversion Control) in any of the three forms: Constructor Injection, Setter Injection, or Interface injection (Frameworks Lecture)

Technical Aspects

- Remember we are using **GitHub** as our versioning control system.
- We are using **Asana** to keep track of user stories, task break down, progress, backlogs, "daily"/frequent standups, and retrospectives.
- Grant access to the TAs.
- There are no limitations regarding frameworks.
- Please use Python
- Javascript is not needed if you develop a simple UI
- Make your project platform independent!!

Submissions

- Submissions are counted at 23:59 of the due date
- If for some reason you don't want to be evaluated on your last commit, make sure you indicate which commit to use (Use the README.md file)
- **Late submission.** We count the last commit before the deadline on the master branch. Late submissions are ignored.

README.md file

- Be verbose in your README file
- Assume the installation is on clean machine, this machine does not have any of your setup. This is not acceptable "it runs and works on my machine". Don't assume facts. Keep track what you installed and include it in the README
- Clear instructions on how to install project dependencies.
- Indicate what parts of the project address the previously agreed tasks with the Product Owner
- Indicate where are you using the **Lecture Aspects** described below
- Clear instructions on how to test and run the project
- Clear instructions regarding additional libraries, packages, db, etc. needed for the system to work (including versions)
- Indicate what frameworks are you using and any other technical aspect needed to review your submission

Including more than one README file is advisable. The system code is commented sufficiently to enable the TA to understand the functionality. Regular stand-up meeting should be agreed upon. Decision making should be documented. Asana should be used for this.

Implementation

The implementation can be a RESTful service or a simple Web UI (frontend).

An example of URI paths for a RESTful service:

URI	HTTP Method	Service	Description
/products	GET	Products	Gets information about all products
/products	POST	Products	Adds a product
/products/{product_id}	GET	Products	Gets a particular product
/products/{product_id}	DELETE	Products	Deletes a particular product
/products/{product_id}	PUT	Products	Updates a particular product
/customers	GET	Customers	Gets information about all customers

For example, something like this would add ProductID 2 to customer 123 with 5 of those with ID 2

```
POST /customers/123/basket
{
"ProductId": "2"

"Quantity": "5"
}
```

Many of your tasks will fall under CRUD: Create, Read, Update, and Delete for the services.